第 3 节 VHDL 语言的常用语法

[学习要求] 掌握 VHDL 硬件描述语言的基本描述语句。并可以利用这些语句进行简单电路的设计。

[重点与难点]

重点：常用的并行语句与顺序语句的语法。

难点：部件（Component 的定义与应用。

[理论内容]

一、并行语句

所谓的并行语句指采用这些语法生成的硬件电路在时间上可以并行（或并发）的执行（运行）。这是 VHDL 语法必须具备的能力，也符合硬件电路的特性。这一点不同于软件，因为软件的语句（或指令）一般总是顺序执行。基本的并行同时语句，可分为下面三种形式来讨论：直接设置语句、条件式信号设置语句和选掼式信号设置语句。

1、直接设置语句

直接设置语句是采用“<=”运算符。

例如如下的语句：

```
D<= not A;
E<=B and C;
F<=A or B or C;
```

这三条语句虽然是分三行写的，但实际上三条语句是同时执行的。

2、条件式信号设置语句： When-Else

When-Else 命令也是属于同时并行的语句命令，它的语法格式如下：

```
信号 A  <=  信号 B When（条件 1） Else
            信号 C When（条件 2） Else
            信号 D;
```

说明：

⑴ 上述的条件式，是指一般常见的布尔表达式，亦即条件式的结果必定是真（True）或错（False）中的一种。

⑵ 语法中的条件式 1 为 True 时，则将信号 B 传递给信号 A，否则再确认条件式 2 为 True 时，将信号 C 传递给信号 A。最后在条件 1 和条件 2 都不成立的情况下，将信号 D 的

值传递给信号 A。

When-Else 命令的应用范围非常广泛，例如：编码器、译码器、多路选择器等的 VHDL 命令编写，都可以采用这条命令。

3、选择式信号设置语句：With-Select

语法格式如下：

With 选择信号 X Select

信号 Y <= 信号值 A When 选择信号 X 值为 m，

信号值 B When 选择信号 X 值为 n

。。。

信号值 Z When Others；

说明：

⑴ With-Select 的命令作用是，判断选择信号 X 的值，依次是 m 或 n 等的相应条件值，然后在判断成立时，将它对应的信号值 A 或信号值 B 传递给信号 Y。

⑵ 而在比罗过程，选择信号 X 无一是上述表示的信号值时，最后会将 Others 保留字前的信号值 Z 传递给信号 Y。

⑶ 上述 With-Select 语法命令的 m，n 等值，必须互不相同。

举例：分别用 With-Select 和 When-else 语句实现以下的真值表。

真值表（True Table）

输 入		输出
X	Y	Z
0	0	0
0	1	1
1	0	1
1	1	0

VHDL 程序如下：

```
--***************库定义部分*************
Library ieee;
Use ieee.std_logic_1164.all;
Use ieee.std_logic_arith.all;
```

```
Use ieee.std_logic_unsigned.all;
--**************实体部分******************
Entity True_Table is
    Port(
        X,Y : in std_logic;
        Z : out std_logic
        );
end True_Table;
--*************用 When-else 语句实现的结构体*****************
architecture a of True_Table is
begin
    Z<= ’0’ when  (X=’0’ and Y=’0’) Else
        ’1’ when  (X=’0’ and Y=’1’) Else
        ’1’ when  (X=’1’ and Y=’0’) Else
        ‘0’;
end a;

--*************用 With-Select 语句实现的结构体*****************
architecture b of True_Table is
    signal S : std_logic_vector(1 downto 0);
begin
    S<=X&Y;
    with S Select
        Z<= ’0’   when “00”,
            ‘1’   when “01”,
            “1’   when “10”,
            ‘0’   when others;
end b;
```

4、Process 语句

Process（进程）语句是一种并行处理语句，在一个构造体中多个 Process 语句可以同时

并行运行。因此，Process 语句是 VHD 语言中描述硬件系统并行行为的最基本的语句。其语法结构如下：

[进程名]： Process （信号 1，信号 2，…）

Begin

….

End Process;

进程 Process 语句中总是带有 1 个或几个信号量。这些信号是 PROCESS 的输入信号（不一定是所有的输入信号），在 VHDL 语言中也称为敏感量。这些信号无论哪一个发生变化（如由“0”变“1”或由“1”变“0”）都将启动该 Process 语句。一旦启动以后，Process 中的语句将从上到下逐句执行一遍。当最后一条语句执行完毕以后，就返回到开始的 Process 语句，等待下一次变化的出现。当没有敏感量时，进程将无限循环执行。

二、顺序语句

所谓的顺序语句如前面所提到的，就是语句是按先后顺序执行的。顺序描述语句只能出现在并行语句 Process 中。

1、If-else 语句

语法格式：

if 条件式 1 Then

语句命令方块 A

Elsif 条件式 2 Then

语句命令方块 B

。。。

Else

语句命令方块 N

End if；

说明：

⑴ 上述命令的动作方式是，先判断条件 1 结果是否为 True，为 True 则执行语句命令方块 A，执行完方块 A 的命令后，则跳至 End if 之后的命令继续执行。若条件 1 结果是 Flase，则往下判断条件 2 为 True 后执行命令方块 B，假设没有任何一个条件成立，则最后执行命令方块 N。

⑵ 上述的 If-else 语句是最完整的写法，它还可有有几种变形：比如

```
If 条件式 1 Then
    语句命令方块 A
End if;
```

或：

```
if 条件式 1 Then
    语句命令方块 A
Else
    语句命令方块 B
End if;
```

If-Else 语句和 Process 最常用的作法是下面的程序片段：

```
Process(CP)
Begin
    IF CP'Event AND CP='1' THEN
        .。。。
    END IF;
END Process;
```

这代表通过 Process 与 IF-Else 命令，进行感测“时钟脉冲信号 CP”是否产生变化，而且判断这个变化是否是上升没（由 0 变 1）的变化。

2、Case-When 语句

语法格式如下：

```
Case 选择信号 Is
    When 信号值 1=>
        语句命令方块 A;
    When 信号值 2=>
        语句命令方块 B;
    ….
    When Others=>
        语句命令方块 N;
```

```
        End Case;
```

说明：

⑴ 这个语句与 With-Select 语句相类似，但区别在于是对选择信号的确定值进行选择并执行相应的语句命令方块。

⑵ 语句最后的 When Others 语句不可省略。

关于在并行语句中提到的真值表也可以用 If-Else 语句和 Case-When 语句来书写，程序片段如下：

```
--**********用 If-Else 语句实现的结构体**********
architecture c of True_Table is
        signal S : std_logic_vector(1 downto 0);
begin
    S<=X&Y;
    Process
    Begin
        If S=”00” then
            Z<=’0’;
        Elsif S=”01” then
            Z<=’1’;
        Elsif S=”10” then
            Z<=’1’;
        Else
            Z,=’0’;
        End if;
    End process;
end c;

--**********用 Case-When 语句实现的结构体**********
architecture d of True_Table is
        signal S : std_logic_vector(1 downto 0);
```

```
begin
    S<=X&Y;
    Process
    Begin
        Case s is
            When “00”=>
                Z<=’0’;
            When “01”=>
                Z<=’1’;
            When “10”=>
                Z<=’1’;
            When others =>
                Z<=’0’;
        End case;
    End process;
end d;
```

三、部件的定义和映像

当电路中存在重复的部分电路，如果考虑不重复使用，就必须重新编写一次，如此一来设计可能常常要做重复的工作，而电路开发时间却因此加长许多。事实上 VHDL 语言提供了部件定义（Component）、部件映像（Port Map）来解决这样的问题。下面是这两个命令的语法格式：

Component 语法格式：

```
Component  部件名称
    Port（
            信号 A ：  端口模式     数据类型；
            信号 B ：  端口模式     数据类型；
            ….
            ）；
End Component；
```

Port Map 语法格式一： 使用=>符号映像部件信号与输入信号之间的关系：

```
Port Map (
        部件信号 A=> 信号 A1；
        部件信号 B=> 信号 B1；
        …..
        );
```

Port Map 语法格式二： 直接由输入信号的关系位置作映像：

部件标题：部件名称 Port Map（信号 A1，信号 B1，….）；

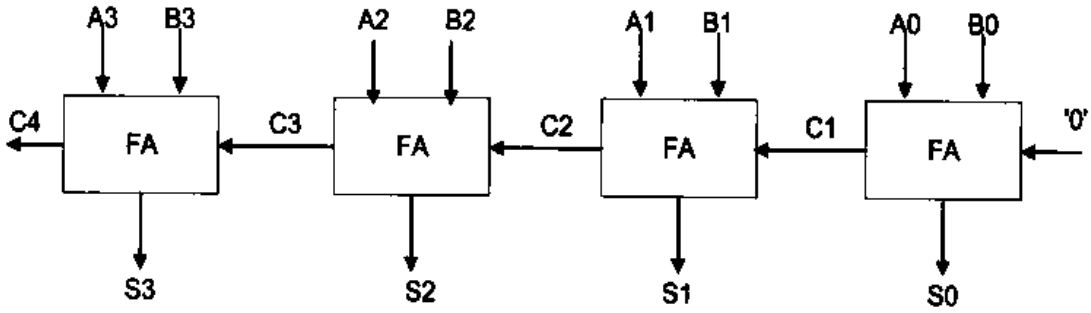

图 1 由全加器组成的 4 位加法器

举例：以一个传统的 4 位加法器为例，这可由 4 个全加器来组成，如图 1 所示。

下面先设计一个全加器，然后再尝试用 Component、Port Map 语法，分别作部件定义和 4 次的部件映像，以便设计出图 1 的 4 位加法器。

首先设计一个全加器的 VHDL 程序，假设名称是 FullAdder.VHD。而这个 FullAdder 的 VHDL 程序实体如下：

```
ENTITY FullAdder is
    PORT(
        A       : IN    std_logic;
        B       : IN    std_logic;
        C       : IN    std_logic;
        Carry   :OUT    std_logic;
        Sum     :OUT    std_logic
        );
END FullAdder;
```

依照上述 FullAdder 程序的管脚，在设计图 1 的 VHDL 程序里的信号定义区中使用 Component 命令定义这个全加器，命令如下：

```
Component FullAdder is
        PORT(
            A       : IN    std_logic;
            B       : IN    std_logic;
            C       : IN    std_logic;
            Carry   :OUT    std_logic;
            Sum     :OUT    std_logic
            );
    END Component;
```

再使用 4 次 Port Map 映像这个加法器，来完成图 1 的 4 位加法器，命令如下：

```
U0：Fulladder Port Map(A(0),B(0),C(0),C(1),S(0));
U1：Fulladder Port Map(A(1),B(1),C(1),C(2),S(1));
U2：Fulladder Port Map(A(2),B(2),C(2),C(3),S(2));
U3：Fulladder Port Map(A(3),B(3),C(3),C(4),S(3));
```

关于 VHDL 语言还有很多其他的语法，这里不再做介绍，大家可以参考一些专门的书籍。